RADC-TR-90-25
Final Technical Report
April 1990

MACHINE LEARNING

The MITRE Corporation

Melissa P. Chase

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Air Development Center
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-90-25 has been reviewed and is approved for publication.

APPROVED: Mark T Maybury

MARK T. MAYBURY, 1LT, USAF
Project Engineer

APPROVED: Raymond P. Urtz Jr.

RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER: Igor G. Plonisch

IGOR G. PLONISCH
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COES) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OPM No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1990	Final Oct 88 to Sep 89

4. TITLE AND SUBTITLE
MACHINE LEARNING

5. FUNDING NUMBERS
C - F19628-89-C-0001
PE - 62702F
PR - MOIE
TA - 79
WU - 80

6. AUTHOR(S)
Melissa P. Chase

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
The MITRE Corporation
Burlington Road
Bedford MA 01730-0208

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Air Development Center (COES)
Griffiss AFB NY 13441-5700

10 SPONSORING/MONITORING AGENCY REPORT NUMBER
RADC-TR-90-25

11. SUPPLEMENTARY NOTES
RADC Project Engineer: Mark T. Maybury, 1LT, USAF/COES/(315) 330-3655

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This report summarizes the design and implementation of an intelligent planner that improves its own performance as it solves problems. The system, called ULS, is based upon explanation-based learning techniques, but addresses some of the weaknesses in that technology that becomes particularly apparent as it is applied to realistic problems.

14 SUBJECT TERMS
Artificial Intelligence, Learning, Planning

15 NUMBER OF PAGES
24

16. PRICE CODE

17 SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19 SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

SECTION 1
INTRODUCTION

1.1 BACKGROUND

Research in machine learning has taken two directions in the problem of acquiring concept descriptions from examples: inductive learning and explanation-based learning. Typically, inductive learning algorithms examine multiple positive and negative examples to determine which features are present in a description of the target concept. Explanation-based learning algorithms, on the other hand, use a domain theory to analyze a single example and construct an explanation of why that example is a member of the target concept; this explanation is then used to form a concept description[Mit86, DeJ86, Ell89].

Explanation-based learning has received a great deal of attention during the past few years. It has appealed to researchers because it captures our intuitions that it is possible to learn a great deal from a single example. One of the major applications of explanation-based learning techniques has been to improve the performance of problem solvers through the acquisition of search control knowledge. Researchers had assumed that using learned search control knowledge would necessarily improve a problem solver's performance, but recently evidence that casts doubt on this assumption has been reported for both SOAR[Tam88] and STRIPS-like problem solvers[Min85, Min88]. This empirical evidence suggests that for machine learning techniques to genuinely improve a system's performance, the *utility* of what is learned must be taken into consideration. For, if the cost of testing the applicability of the search control knowledge is greater than the savings realized by reducing the search, the learned search control knowledge may actually degrade the problem solver's performance.

1.2 OBJECTIVES

Our goal is to design and implement a planner that improves its own performance as it solves problems. This system, called ULS, is based upon explanation-based learning techniques, but addresses some of the weaknesses in that technology that become particularly apparent as it is applied to realistic problems. Our research has been specifically directed toward the above-mentioned utility problem. Our approach to addressing the utility of learning is to approximate the results of explanation-based learning[Cha89, Zwe88].

This year, our efforts were focused on improving the approximation techniques, defining new simplified domains, and conducting experiments in various domains.

1.3 ORGANIZATION OF THIS DOCUMENT

This paper describes ULS at the end of FY89. In Section 2, we present the problem of acquiring useful search control knowledge in a more general setting, namely, the familiar theme of viewing learning search control knowledge as a search process itself. We describe some previous solutions to this problem within this setting, as well as place our own approach within this framework. In Section 3, we describe the implementation of ULS. In Section 4, we discuss the experiments that we conducted this year. In Section 5, we discuss our results and some future directions.

SECTION 2
FRAMEWORK

2.1 ACHIEVING UTILITY THROUGH SEARCH

Mitchell formulated the task of learning a concept description as a search process[Mit82]. More recently, Keller has extended this idea to the problem of learning an approximate concept description, that is, the description may sacrifice accuracy to achieve some performance objectives[Kel88]. We would like to elaborate upon this framework in the context of acquiring *useful* search control knowledge. To see how this particular task may be viewed as a state space search, we must specify the following features common to all search problems:

1. The states of the search space.
2. The initial state.
3. The search operators, i.e., state transition functions.
4. The search termination condition, i.e., the objectives of the search.
5. The heuristic guidance of the search.

In our search control acquisition problem, the search space consists of sets of search control concept descriptions. Each concept description represents a search control heuristic that guides the problem solver's search, while each set of concept descriptions is a collection of such heuristics.[1] The search begins with an initial hypothesis set of descriptions. Search operators navigate through the space by applying transformations to a set of concept descriptions as a whole, or to a single heuristic within the set. Some of these operators obey the subset relation[Sim88], that is, the set of instances (the situations in which the heuristics are applicable) covered by the set of concept descriptions after the operator is applied is a subset of the set of instances covered by the old set of concept descriptions, or *vice versa*. When the new set is a subset of the old set, the operator is a specializing transformation; when the old set is a subset of the new set, the operator is a generalizing transformation; and when each set is a subset of the other, the operator is a truth-preserving transformation. Operators which do not obey the

[1] We must always keep in mind that there are two searches: (1) the search carried out by the problem solver, or performance element of the system; and (2) the search carried out by the learning component which acquires heuristics to guide the first type of search. The latter is the case under consideration in this paper.

subset relation are called approximating transformations. After such an operator is applied, the old and new sets of instances intersect, but the new set may contain negative instances that had been excluded by the old concept descriptions and omit positive instances that had been covered by the old set of concept descriptions. The search objectives are met when the collection of rules satisfies some utility criterion. Since the search objectives may involve performance criteria as well as accuracy criteria, the collection of rules may only approximate the target search control concepts in the sense that some instances may have been misclassified (i.e., a rule may cover a situation that it should not, or a rule may not cover a situation that it should).[2] The search can be guided by a variety of heuristics, such as domain knowledge and performance measures.

2.2 PREVIOUS WORK

Minton's PRODIGY[Min88, Min87] and Keller's MetaLEX[Kel87b, Kel87a] are two systems that address the utility of learning. Before describing our own approach, which is strongly influenced by these two systems, we would like to reconstruct both of them within the framework outlined above.

In PRODIGY the search for useful search control heuristics begins with an initial hypothesis, namely a search control rule, that has been acquired through explanation-based learning. PRODIGY then applies truth-preserving search control operators to this rule. These operators compress the rule by applying logical identities and domain-dependent simplifications. The search objective is to maximize the average utility, defined by

$$(\textit{average-savings} \times \textit{application-frequency}) - \textit{average-match-cost}$$

In practice, the utility of a search control rule is estimated from the training instance by empirically measuring the match cost of the rule and by measuring the time spent exploring the portion of the tree that would have been eliminated by the rule. Although PRODIGY considers performance as an element of its search objectives, since it applies only truth-preserving operators, it acquires search control rules that cover the same instances as the initial hypothesis. In PRODIGY, the search for control rules is guided by domain-dependent and domain-independent knowledge, as well as by performance measures.

In MetaLEX the search begins with an initial "useful" concept description acquired from an inductive learner. The search operators which may be applied

[2] An approximate concept description may be generated by an operator that obeys the subset relation. For example, we may apply a generalization operator so that the more general description now includes negative instances that had been excluded by the more specific description.

to a concept description are Truify and Falsify, which replace subexpressions in a concept description with T (the atom T) or F (the atom NIL), respectively. Truify generalizes and Falsify specializes a concept description. In MetaLEX the search objectives are satisfied when performance is improved as measured by efficiency (cpu time is less than some threshold) and effectiveness (the number of problems solved is greater than some minimum number). Since MetaLEX both considers performance in its search objectives and applies non-truth-preserving transformations, it acquires concept descriptions that approximate the original concept description. The search is guided by monitoring the system's performance on a set of training examples.

2.3 OUR APPROACH

Our chief goal is to build a system that will be able to improve its performance over time without the need for a set of training examples. We are developing a system called ULS to achieve this goal by combining both analytical and empirical techniques. As we describe ULS within the search framework its debts to both PRODIGY and MetaLEX will be apparent.

In ULS the search for useful search control knowledge begins with an initial hypothesis description, i.e., a search control rule, acquired through explanation-based learning. The operators that navigate through the space of sets of search control rules are generalizing and specializing transformations, as in MetaLEX. The objective of this search is to satisfy a measure of utility similar to that used in PRODIGY. In practice we estimate this utility by gathering statistics on rule applications to suggest which generalizations and specializations will reduce the cost of rule application without unduly diminishing the accuracy of the rule. Again, as in MetaLEX, since ULS considers performance in its search objectives and applies non-truth-preserving operators, it acquires search control rules that approximate the original rule. The search is guided by by monitoring the system's performance during ordinary problem solving.

The major contribution of this research, and the one that will be emphasized in the following section, is the way ULS estimates the utility of rules and how it guides its search.

SECTION 3
IMPLEMENTATION

ULS consists of a problem solver, an explanation-based learner, and a rule-transformer.

3.1 PROBLEM SOLVER

The ULS problem solver is a STRIPS-like problem solver[Fik72]. Tasks are given to the problem solver in the form of a conjunction of predicates. The system attempts to find a sequence of operator applications that will change the world state so that the conjunction of goals is true in the changed world state. Each of these operators consists of a conjunction of preconditions that determines if an operator is applicable, and an add-list and a delete-list that describe the condition of the world state after the operator is applied. The problem solving process is one of search. During each decision cycle, the problem solver first selects a goal to satisfy and then selects an operator that makes the goal true. When an operator's preconditions are not true, they become goals that must be satisfied.

The problem solver's search can be guided by applying rules that prefer or reject a search control decision. Our experiments have focused upon operator preference rules. In this case, when an operator must be selected, all operator preference rules are evaluated and those that evaluate to true vote to prefer an operator. The operator with the most votes is the one that is applied.

When there are no search control rules, or when the application of preference rules produces a tie, ULS follows some arbitrary search strategy. Initially, this strategy was depth-first search. Later, we employed random search as the default strategy. We did this so that the explanation-based learning component would be able to acquire search control rules more readily. With depth-first search, many operators were never tried (since it was too expensive in time and space to completely explore the search space) and hence were never candidates for subjects of search control rules.

3.2 EXPLANATION-BASED LEARNER

ULS's explanation-based learner is modeled on PRODIGY. Knowledge about the problem solver's architecture, as well as knowledge about the application domain, are represented as schemas.

The architecture-level schemas describe the behavior of the problem-solver; in particular they define what it means for choices at the various points in the decision cycle to be successful or unsuccessful. For example, the schemas that describe the search control concept of a successful operator choice are:

```
(OPERATOR-SUCCEEDS-SCHEMA-1
     :comment     ''An operator succeeds if it directly
                  solves the problem''
     :if-part     (and  (added-by-operator ?goal ?op)
                        (operator-applicable ?op ?node))
     :then-part   (operator-succeeds ?op ?goal ?node))
```

and

```
(OPERATOR-SUCCEEDS-SCHEMA-2
     :comment     ''An operator succeeds if it succeeds
                  at a descendant of node''
     :if-part     (and  (operator-succeeds-at-descendant
                              ?op ?goal ?node ?descendant)
                        (regress-goal ?goal ?descendant ?node)
                        (regress-preconditions ?descendant ?node))
     :then-part   (operator-succeeds ?op ?goal ?node))
```

The application-level schemas encode information about the actions in the domain. This knowledge is of two varieties: information about the applicability of an operator and information about the effects of an operator. Both sorts of knowledge are derived automatically from the operator descriptions.

For example, the schema that describes what it means for the operator goto-box to be applicable, which is derived from the operator's preconditions, is:

```
(OPERATOR-APPLICABLE-SCHEMA-1
     :if-part     (and  (operator-at-node ?op ?node)
                        (matches ?op goto-box)
                        (known (and (type ?box box)
                                    (inroom ?box ?room)
                                    (inroom robot ?room))))
     :then-part   (operator-applicable ?op ?node)
     :domain      robot-world
```

and the schema that describes effects of the operator goto-box, which is derived from the operator's add-list, is:

```
(ADDED-BY-OPERATOR-SCHEMA-1
     :if-part    (and (matches ?op goto-box)
                      (matches ?goal (nextto robot ?box))
     :then-part  (added-by-operator ?goal ?op)
     :domain     robot-world
```

The explanation-based learning algorithm used in ULS is essentially schema specialization. The learner is given a target concept, such as,

```
(operator-succeeds ?op ?goal ?node)
```

and a training instance drawn from the search tree constructed by the problem solver during a problem-solving session, such as

```
(operator-succeeds push-box-to-door
                   '(nextto box3 door1-2) node-3).
```

ULS constructs an explanation of why the training instance is an example of the target concept by progressively specializing schema in the context of the search tree. The explanation is then converted into a search control rule:

```
(OPERATOR-PREFERENCE-RULE-1
     :if-part    (and  (current-node ?node)
                       (is-goal ?node ?goal)
                       (matches ?goal '(nextto ?var86 ?var82))
                       (matches ?op push-box-to-door)
                       (known '(connects ?var82 ?var83 ?var84) ?node)
                       (known '(inroom robot ?var83) ?node)
                       (known '(type ?var84 office) ?node)
                       (known '(status ?var82 open) ?node)
                       (known '(type ?var82 door) ?node)
                       (known '(connects ?var82 ?var84 ?var83) ?node)
                       (known '(inroom ?var86 ?var84) ?node)
                       (known '(pushable ?var86) ?node)
                       (known '(type ?var86 box) ?node))
     :then-part  (prefer ?op))
```

This year, in order to reduce the time and space used during learning, the schemas have been proceduralized.

3.3 RULE TRANSFORMER

The rule transformation component of ULS produces rules that approximate the search control rules acquired through explanation-based learning. It applies generalization and specialization operators in order to transform the rules into more useful ones, which are now efficient to test and still accurately reduce the problem solver's need to search. The generalization operator drops conditions in the search control if statistically justified, while the specialization operator adds conditions that had been previously dropped.

3.3.1 Justifying Generalization

Ideally, one wants to base the decision to apply the generalization operator upon the utility of the search control rule obtained by applying these operators. In practice, it is usually difficult to carry out a utility analysis, so we estimate the utility. Since the generalization operator drops conditions, the resulting rule usually will be less expensive to evaluate. To ensure that dropping a condition will not lead to greater search, we only drop conditions that seem to be predicted by other conditions in the rule; hence we may assume that this condition is superfluous and its presence or absence will not affect the truth value of the rule's conditions.

This estimation procedure is implemented as follows. ULS's rule evaluator tests conditions from left to right, so we compute the conditional probability that a condition is true given that the conditions to its left are true.[3] Symbolically, if the antecedent of a rule is a conjunction of conditions, $C_1 \wedge C_2 \wedge \cdots \wedge C_n$, the conditional probability that C_k is true given that $C_1, C_2, \ldots, C_{k-1}$ are true is

$$P(C_k) \mid \bigcap_{i=1}^{k-1} P(C_i)) = \frac{\bigcap_{i=1}^{k} P(C_i)}{\bigcap_{i=1}^{k-1} P(C_i)}$$

When a rule is applied, tallies are kept of the number of times a condition is tested and the number of times the condition is true. Since a condition is never tested unless the conditions to its left are true, the ratio

$$\frac{\textit{number of times condition is true}}{\textit{number of times condition is tested}}$$

gives the desired conditional probability. When a condition's conditional probability exceeds a pre-set threshhold, θ, and the condition has been tested a sufficient number of times, the condition may be dropped, provided it does not

[3] If rules are evaluated in some other fashion, different conditional probabilities would have to be computed, but the general idea of capturing redundant conditions in this way would still be appropriate.

introduce variable bindings. Bernoulli's Theorem[Pea84] is used to determine the number of tests needed to guarantee with sufficiently high probability (η) that the difference between the actual conditional probability (π) and the sample conditional probability (ρ) will be less than ϵ:

$$n = \frac{2}{\epsilon^2} \log \frac{2}{1-\eta}$$

Even when the threshhold, θ, is surpassed and the number of tests determined by Bernoulli's Theorem is satisfied, a condition might not be dropped for one of two reasons.

First, a condition may have unbound variables and serves as a generator for those variable bindings. Whenever there are no constraints on the potential bindings (e.g., all the variables in the condition are unbound), every time the condition is tested, it will match against the world state, generate a set of bindings, and succeed. In this case the conditional probability for this condition would be 1.0. Even when some variables in the condition are bound, the condition is likely to match against the world state and succeed. Consequently, we have chosen to not drop conditions that generate variable bindings.

The second situation results from the effect dropping a condition has on the conditional probabilities of other conditions. When a condition is dropped, the tallies representing the conditional probabilities of all conditions to its right are invalidated (since they depend in part on the dropped condition) and are reset to 0. If the conditional probability of one of those conditions is over the threshhold, θ, and the number of tests is "close" to the required number, a lot of work would be lost by zeroing the tally. To circumvent this, ULS does not drop a literal when ones to its right are close to being dropped. A literal is close to being dropped if

$$n - t \leq a \cdot p \cdot \Delta t,$$

where n is the number of tests required (derived from Bernoulli's Theorem), t is the number of tests already made, a is the average number of times a rule is evaluated per problem, p is the number of problems we are willing to wait before dropping, and Δt is the rate of change of the number of times a rules is tested per problem.

For the operator preference rule given above, the following statistics were gathered:

Condition	*# Tests*	*# True*	*Prob*
(known '(connects ?var82 ?var83 ?var84) ?node)	763	763	1.00
(known '(inroom robot ?var83) ?node)	763	458	0.60
(known '(type ?var84 office) ?node)	458	458	1.00
(known '(status ?var82 open) ?node)	458	403	0.88
(known '(type ?var82 door) ?node)	403	403	1.00
(known '(connects ?var82 ?var84 ?var83) ?node)	403	403	1.00
(known '(inroom ?var86 ?var84) ?node)	403	403	1.00
(known '(pushable ?var86) ?node)	403	270	0.67
(known '(type ?var86 box) ?node)	270	270	1.00

With θ set to .8 and the number of tests set to 267 (for $\epsilon = .15$ and $\eta = .95$), the following rule was created by dropping those conditions eligible to be dropped:

```
(OPERATOR-PREFERENCE-RULE-1
     :if-part    (and  (current-node ?node)
                       (is-goal ?node ?goal)
                       (matches ?goal '(nextto ?var86 ?var82))
                       (matches ?op push-box-to-door)
                       (known '(connects ?var82 ?var83 ?var84) ?node)
                       (known '(inroom robot ?var83) ?node)
                       (known '(pushable ?var86) ?node))
     :then-part  (prefer ?op))
```

All conditions whose conditional probability was greater than .8 and had been tested more than 267 were dropped, except for `(known '(connects ?var82 ?var83 ?var84) ?node)`. It was not dropped because it is a generator for two unbound variables, `?var83` and `?var84`.

3.3.2 Justifying Specialization

As with generalization, one would like to base the decision to specialize upon a utility analysis. Again, ULS estimates the utility by relying upon statistics gathered during problem solving. If an approximated rule applies inappropriately, that is, the rule suggests applying an operator and that choice leads to backtracking, the approximation may be the culprit. In order to determine if this is the case, the original rule is evaluated to see if it would have voted differently from the generalized rule. If that happens a sufficient number of times,

ULS decides that the generalization is responsible for the misapplications of the rule, and then employs some strategy to determine which missing condition or conditions is to blame. The first strategy we have implemented is to test each dropped condition individually. For example, if after the above approximated rule was created, doors are now closed more often than open, this rule may misapply a sufficient number of times to be blamed. When it is blamed, `(known '(status ?var82 open) ?node)` will be the only dropped condition that has a different value, so it will be restored.

The specialization component has been designed and a prototype has been implemented, but not yet tested. One problem we encountered was that the domains we have been using are not suitable for specialization (as described above). For specialization to be useful, the domain must have several operators which can achieve the same goal but have different sets of preconditions, some of which cannot be made true. This characteristic causes backtracking to occur; otherwise the first operator tried will eventually succeed through subgoaling. We have just recently implemented some domains which will be suitable for testing specialization.

SECTION 4
EXPERIMENTS

We have been conducting experiments using a two-room and a three-room STRIPS domain[Fik72], and an extended two-room STRIPS domain[Min88] to measure the effectiveness of these techniques. We automatically generate random sets of test problems for a particular configuration of rooms. We can control various parameters, such as the percentage of open doors, in order to introduce regularities into the problems. Although ULS has been designed to interleave learning and problem solving, we decouple these processes when collecting timing data. After using explanation-based learning to learn search control rules and using the techniques described above to approximate these rules, we use a set of 100 problems to run timing tests on the problem solver for three cases: (1) unguided search, (2) search guided by rules acquired through explanation-based learning, and (3) search guided by the same rules after they were approximated.

The first set of tests were carried out in the two-room and three-room STRIPS domain. For each set of problems, two regularities were introduced into the domain: (1) doors are open 90 percent of the time, and (2) boxes are pushable 100 percent of the time. The results demonstrated a modest improvement in performance. In the two-room domain, when given a time limit of 5 minutes, ULS could solve only 23 problems before learning. After learning 27 rules ULS could solve 92 problems, and it solved the same number with the approximated rules (with 52 percent of all conditions dropped). In the three-room domain, ULS could solve only 13 problems before learning. After learning 18 rules ULS could solve 50 problems, and it solved the same number with the approximated rules (with 39 percent of the conditions dropped). The effect of approximation on cumulative plan time for solved problems is summarized in the following table:

Domain	*EBL Rules*	*Approx. Rules*	*% Decrease*
2 Room	4451.22	4382.86	1.53
3 Room	2066.18	2001.07	2.67

As can be seen from the table, the results were not very encouraging. When we profiled the problem solver in order to get a clearer picture of where time was spent during problem solving, we discovered that our timings were ambiguous (in cases where the plan time was greater for the dropped rules than the original rules, with the profiler turned on the plan time for the dropped rules was actually less). We felt that there was some overhead incurred by using PCL (Portable CommonLoops, the Xerox PARC portable implementation of the Common LISP

Object System) which affected the different cases differently, namely in the code that tested whether a condition was dropped (when a predicate is dropped its type is changed to **dropped**; subsequently when a dropped predicate's truth value is tested, the predicate immediately returns **true**). So we decided to run a new set of tests in the three-room STRIPS domain and the extended two-room STRIPS domain. In these tests, dropped predicates were completely ignored.

The three-room STRIPS domain was run on a set of problems similar to those described above. On this set of problems, ULS could only solve 9 problems. After learning 76 rules, ULS could solve 59 problems. After dropping 32 percent of the conditions dropped, ULS solved the same number of problems using the approximated rules. The effect of approximation on cumulative plan time is presented in the following table:

Domain	*EBL Rules*	*Approx. Rules*	*% Decrease*
3 Room	5789.38	5039.40	14.88

In the extended STRIPS domain, boxes can be carried as well as pushed, and doors can be locked. The regularities introduced into the domain were: (1) doors were locked 20 percent of the time, (2) boxes were pushable 100 percent of the time, and (3) boxes were carriable 20 percent of the time. In order to solve any problems, even with learned rules, the time limit was increased to 15 minutes. After learning 129 rules, ULS was able to solve 26 problems within the time limit. After dropping 24 percent of the conditions, ULS was able to solve 27 problems within the time limit. The effect of approximation on cumulative plan time (only for solved problems) is summarized in the following table:

Domain	*EBL Rules*	*Approx. Rules*	*% Decrease*
Extended	5821.54	5255.01	10.08

After reducing the overhead incurred by PCL, the approximated rules demonstrated a clear, although relatively modest, improvement in performance. We will discuss the reasons for these modest results in the following section.

SECTION 5
DISCUSSION

Our approach to acquiring useful search control rules through approximation captures regularities in the application domain and unstated features of the problem solver architecture. By uncovering these relationships, ULS is able to produce a useful set of search control rules. The techniques may be helpful not just for improving the quality of automatically learned rules, but might also improve hand-crafted rules, especially in cases where assumptions made by the knowledge engineer may no longer be valid because of changes in the underlying knowledge base.

In our experiments, we have found that ULS observes and exploits two kinds of regularities in the domain. First, some conditions are almost always true. In the example rule above, the fact that problems were generated so that doors were usually open was captured. Second, some conditions are almost always true conditionally. In the example rule above, the statistics captured the fact that all pushable objects are boxes (since the only objects are `robot` and boxes, and boxes are always pushable).

In addition, we have found that ULS discovers dependencies between conditions in a rule that arise from the problem solver architecture. We see a case of this in the example rule above. Once

```
(known '(connects ?var82 ?var83 ?var84) ?node)
```

is tested, `?var82` is bound to a door. Thus, it is not necessary to test

```
(known '(type ?var82 door) ?node)
```

The relationship between these two conditions arises because the conditions of a rule are tested by unifying them against a database representing the world state.

Our experimental evidence demonstrates that we can use statistical evidence gathered during problem solving to produce approximate search control rules that are still accurate and somewhat more efficient to test. The improvement in performance has not been large because the generalizing and specializing transformations we have used do not really address the combinatorics of rule evaluation. The time to evaluate a rule is $O(n^b)$, where n is the number of conditions, and b is the average number of variable binding choices per condition. Our transformations only reduce the number of conditions and do not affect the number of times a condition is tested (because all variables in those conditions are bound), and hence produce only a linear improvement in performance, which increases as

the number of binding choices increases (e.g., as the complexity of the domain increases). To produce approximate rules that are much more efficient than the original rules, however, it is necessary to reduce the number of binding choices. Thus in the future, we plan to provide ULS with other operators that search the space of search control rules. For example, we would like to apply transformations that reorder the conditions appearing in the left-hand side of a rule, estimating the number of binding choices statistically in order to arrive at a good approximation to the optimal ordering.

LIST OF REFERENCES

[Cha89] Chase, M.P., Zweben, M., Piazza, R.L., Bruger, J.D., Maglio, P.P. Approximating Learned Search Control Knowledge. In *Proceedings of the Sixth International Workshop on Machine Learning*, Ann Arbor, MI, 1989.

[DeJ86] DeJong, G.F. and Mooney, R. Explanation-Based Generalization: An Alternative View. *Machine Learning*, 1, 1986.

[Ell89] Ellman, T. Explanation-Based Learning: A Survey of Programs and Perspectives. *Computing Surveys*, 21, 1989.

[Fik72] Fikes, R.E., Hart, P.E., and Nilsson, N.J. Learning and Executing Generalized Robot Plans. *Artificial Intelligence*, 3, 1972.

[Kel87a] Keller, R.M. Defining Operationality for Explanation-Based Learning. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, WA, 1987.

[Kel87b] Keller, R.M. *The Role of Explicit Knowledge in Learning Concepts to Improve Performance.* PhD thesis, Dept. of Computer Science, Rutgers University, January 1987.

[Kel88] Keller, R.M. Learning Approximate Concept Descriptions. 1988. Unpublished paper.

[Min85] Minton, S. Selectively Generalizing Plans for Problem Solving. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, 1985.

[Min87] Minton, S. and Carbonell, J.G. Strategies for Learning Search Control Rules: An Explanation-Based Approach. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, 1987.

[Min88] Minton, S. *Learning Effective Search Control Knowledge: An Explanation-Based Approach.* PhD thesis, Dept. of Computer Science, Carnegie-Mellon University, March 1988.

[Mit82] Mitchell, T.M. Generalization as Search. *Artificial Intelligence*, 18, 1982.

[Mit86] Mitchell, T.M., Keller, R.M., and Kedar-Cabelli, S.T. Explanation-Based Generalization: A Unifying View. *Machine Learning*, 1, 1986.

[Pea84] Pearl, J. *Heuristics*. Addison Wesley, Reading, MA, 1984.

[Sim88] Sims, M.H., Zweben, M., and Chase, M.P. *An Abstraction of EBL and the Clarification of Approximations*. Technical Report, AI Lab, NASA Ames Research Center, 1988.

[Tam88] Tambe, M. and Newell, A. *Why Some Chunks Are Expensive*. Technical Report, Dept. of Computer Science, Carnegie-Mellon University, 1988.

[Zwe88] Zweben, M. and Chase, M.P. Improving Operationality with Approximate Heuristics. In *Proceedings of the AAAI Spring Symposium on Explanation-Based Learning*, Stanford University, 1988.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic reliability/maintainability and compatibility.